How to Make your own Neural Network (Part-02)

Dr. Jamshaid Ul Rahman

Prepared by: Malik Ahsin Iqbal & Muhammad Israr

ABDUS SALAM SCHOOL OF MATHEMATICAL SCIENCES
Government College University Lahore

Table of Contents:

Link with Part one:

In part one of this presentation it was discussed that how can the matrix for a two layer neural network containing two nodes(neurons) in each layer. The structure of such neural network is given in fig 1. And the equation of matrix is given as

$$X = AI$$

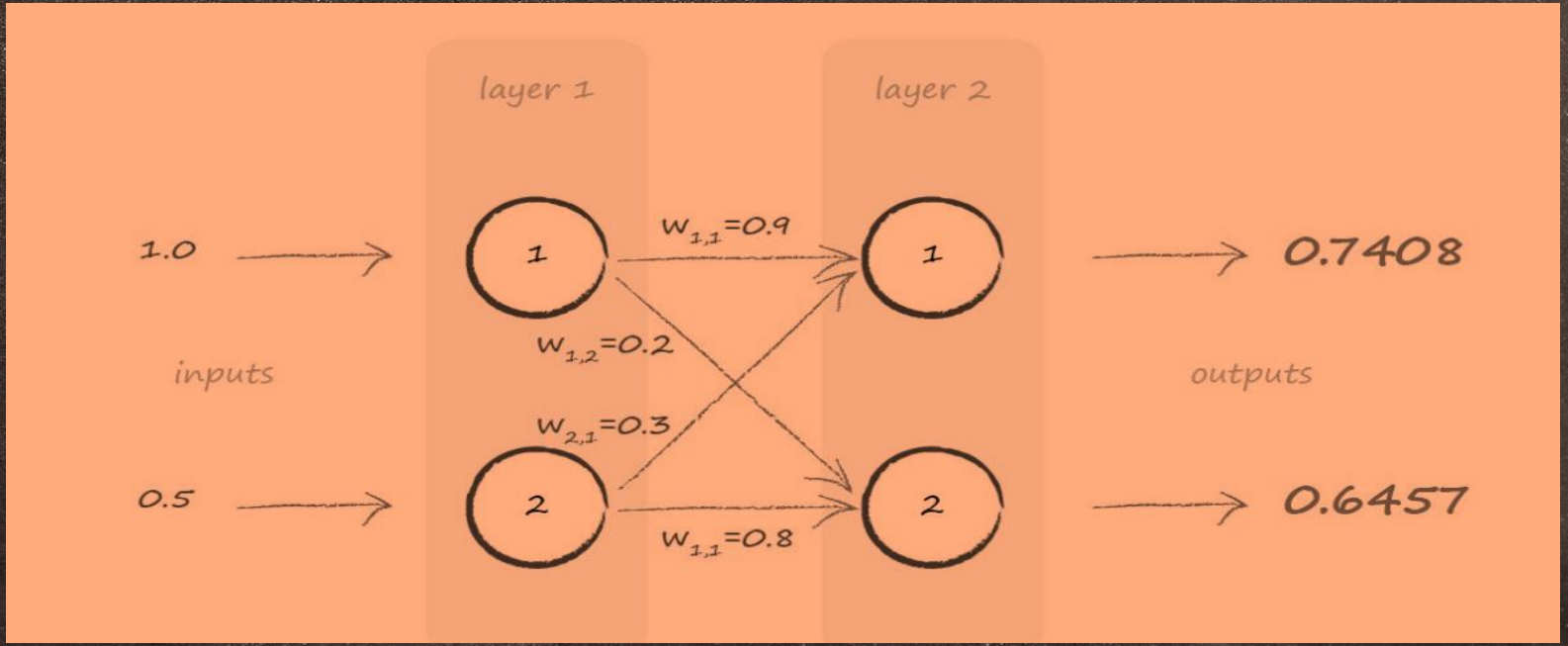
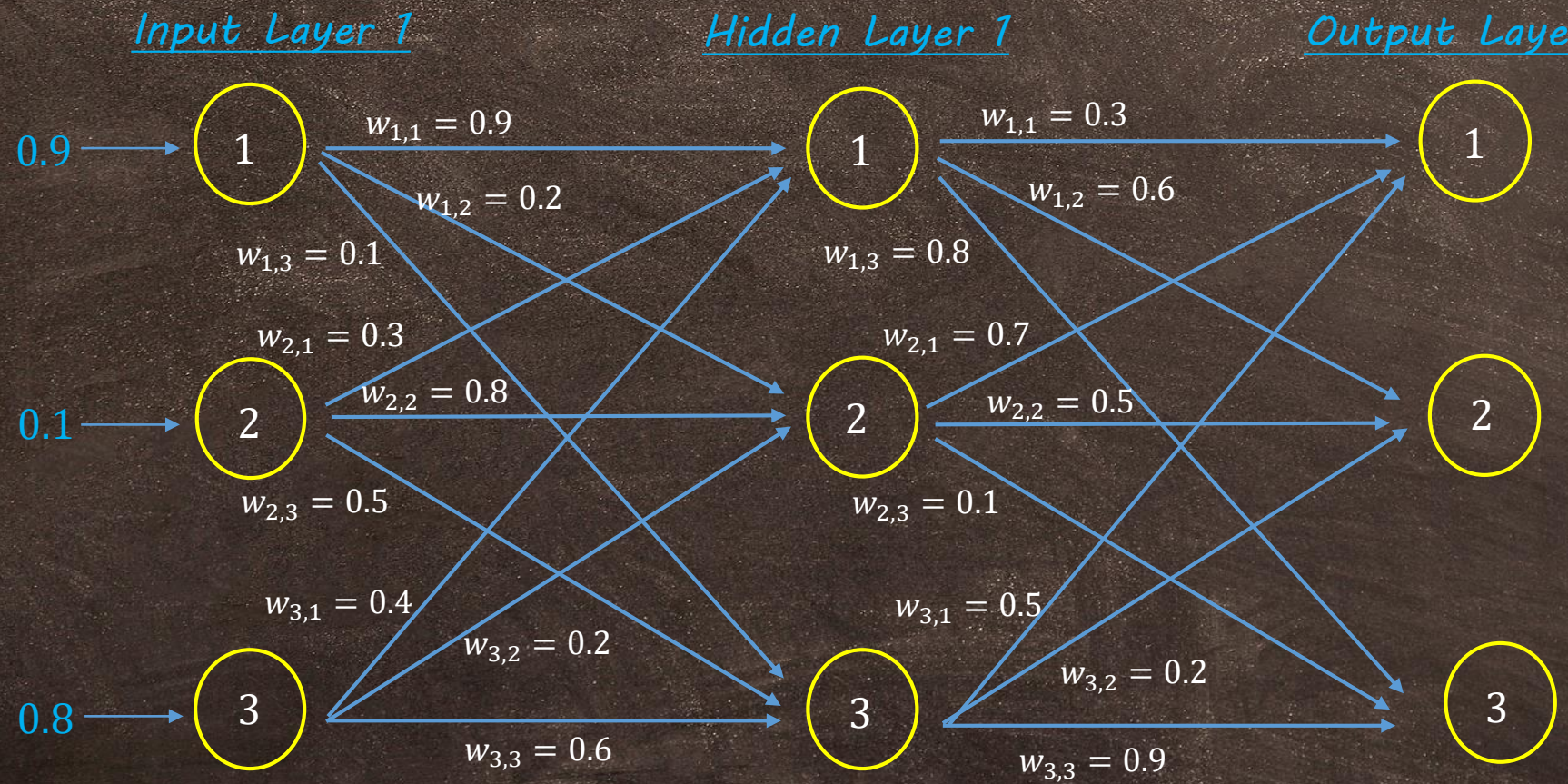


Fig. 1

A Three Layer Example With Matrix Multiplication:

Let us consider the following neural network with three layers having three neurons in each layer.



Malik Ahsin Iqbal

A Three Layer Example With Matrix Multiplication:(cont·)

Matrix Construction:

The input matrix for layer one is given as $I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$

For the input of the hidden layer, we need to work with two matrices, the matrix of inputs from the first layer and then the matrix of weights·

The input matrix is given as $I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$

While the weight matrix is given as

$$W = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{pmatrix}$$

$$W_{input\_hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

A Three Layer Example With Matrix Multiplication:(cont·)

Getting input for hidden layer:

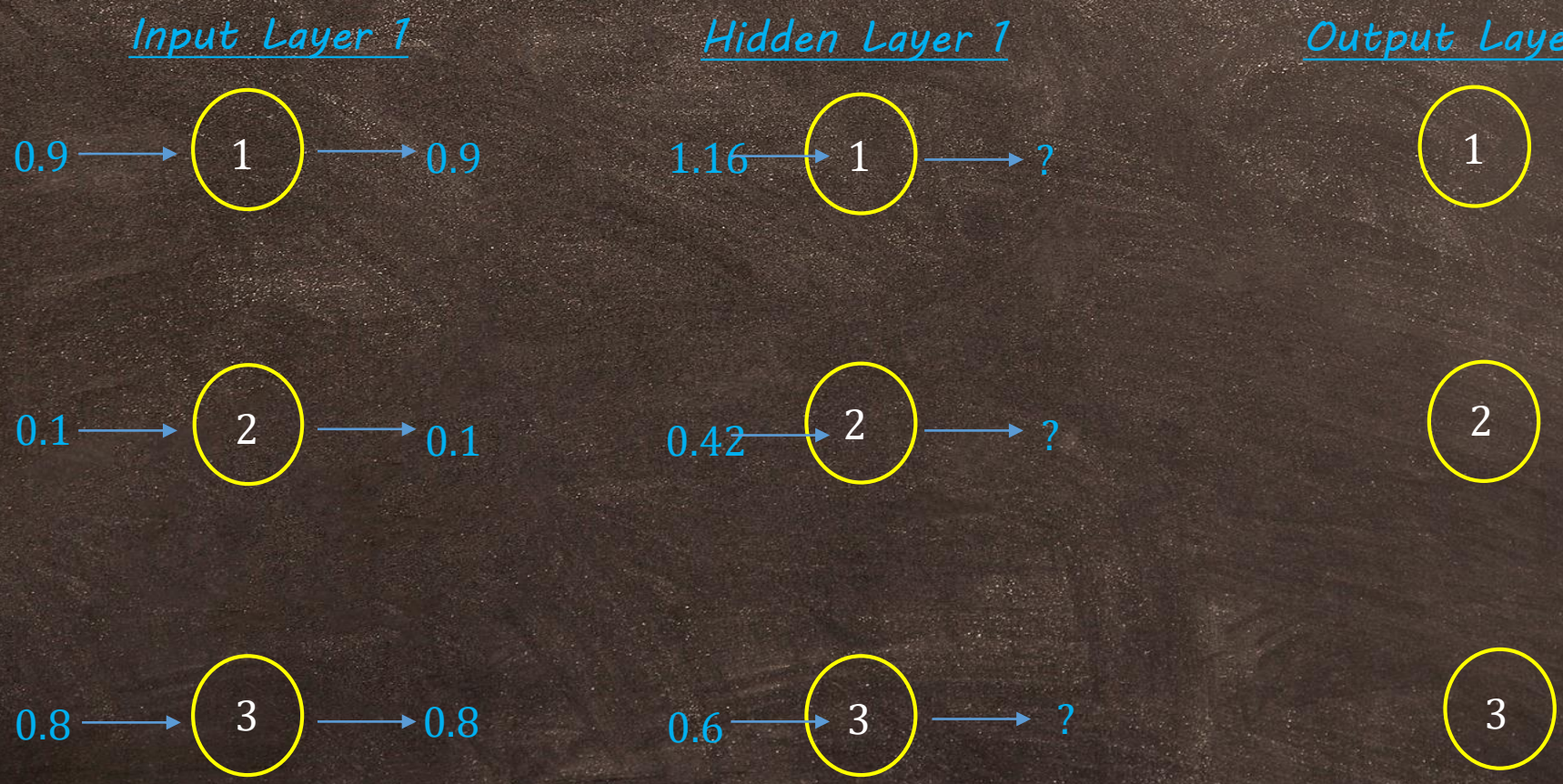
Now we will be using the following method of matrix multiplication tp get the inputs for hidden layer

$$X_{hidden} = W_{input_{hidden}} . I$$

$$X_{hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix} \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$X_{hidden} = \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

A Three Layer Example With Matrix Multiplication:



A Three Layer Example With Matrix Multiplication:

Output from hidden layer:

So far we have reached to the inputs for the hidden layer, now these inputs are going to encounter sigmoid function $y = \frac{1}{1+e^{x}}$, to get the output of this layer.

$$O_{hidden} = Sigmoid \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix} \quad \Rightarrow \quad O_{hidden} = \begin{pmatrix} 0.765 \\ 0.603 \\ 0.650 \end{pmatrix}$$

Now this output from the hidden layer will be operated with the weight matrix between hidden layer and output layer, which is given as :

$$W_{ho} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

A Three Layer Example With Matrix Multiplication:

Input for output layer:

Now for the input of the final layer of our neural network, we need to compute the product between the weight matrix W_{ho} and the output matrix from the hidden layer, which is given as:

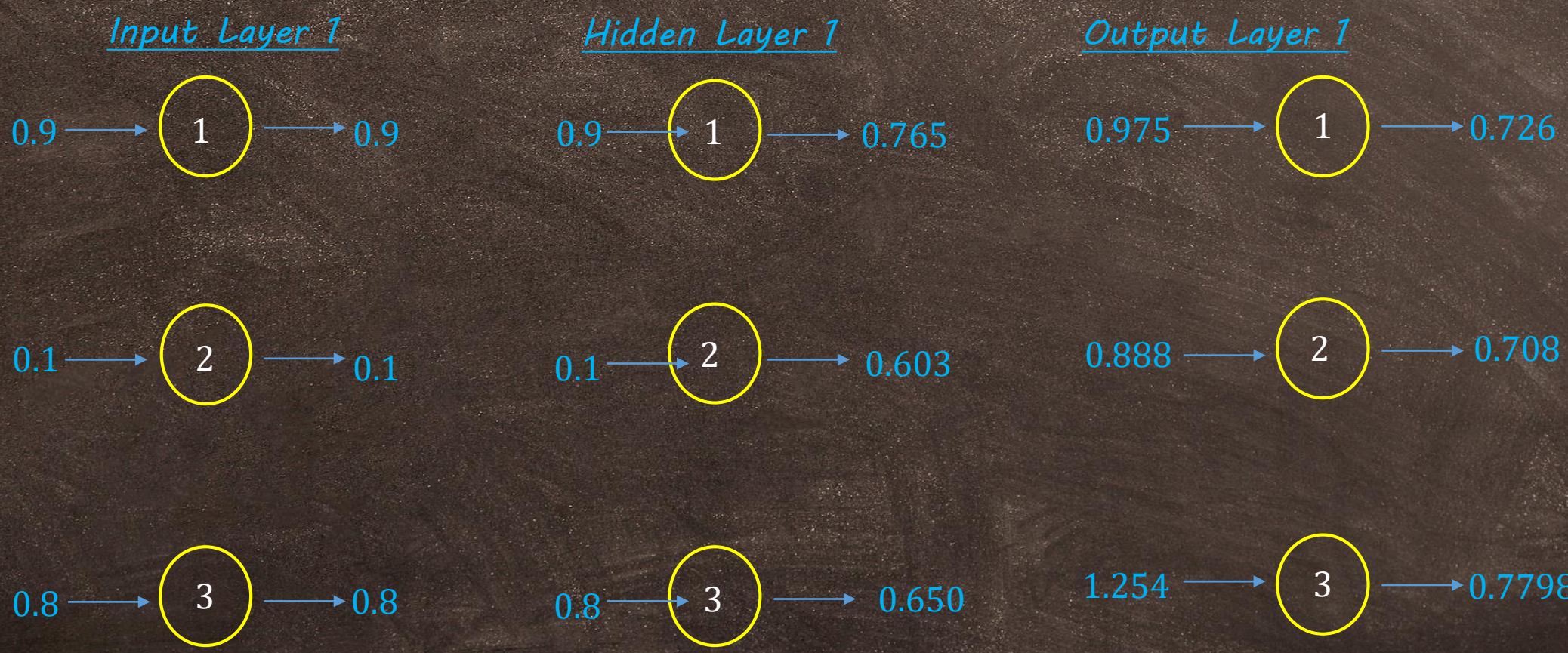
$$I_{output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix} \begin{pmatrix} 0.765 \\ 0.603 \\ 0.650 \end{pmatrix} \Rightarrow \quad I_{output} = \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

And finally, we are in a position to get the output from our neural network which will be computed by operating the sigmoid function over I_{output} matrix.

$$O_{output} = sigmoid\,. \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix} \qquad O_{output} = \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$

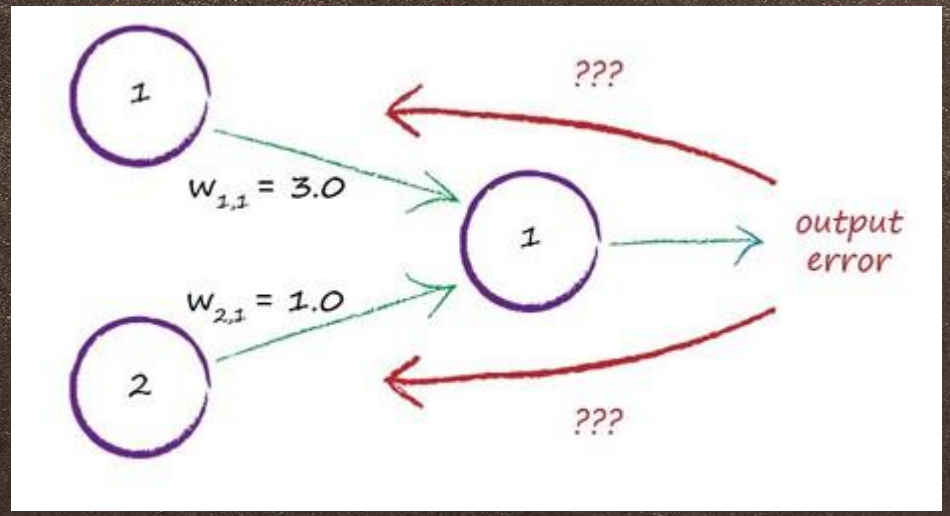
A Three Layer Example With Matrix Multiplication:

Finally the structure of neural network along the inputs and outputs would look like:



Learning weights from more than one node:

Consider thr following example in which two nodes are contributing their weights for the next node. So what would be the output from this node? What would be the output error from this node?

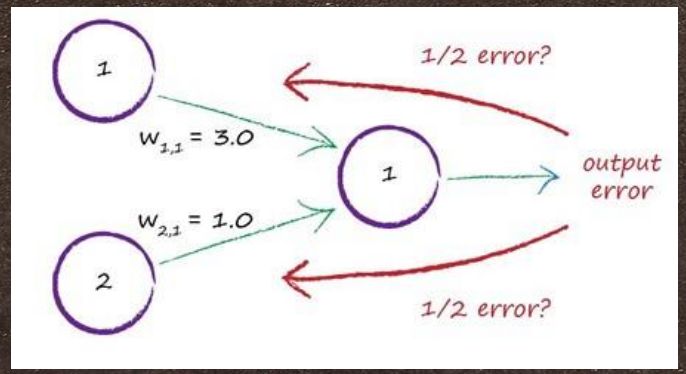


To deal with this issue there are two ideas, first to split the error equally for both nodes and second is to assign the more part of error to the node which has higher weight associate with it.

Learning weights from more than one node:(cont·)

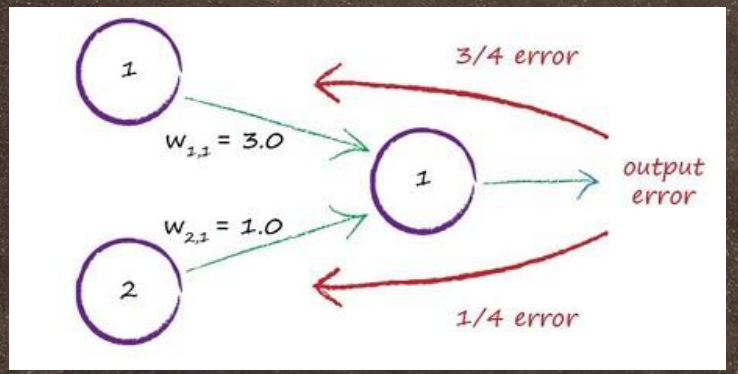
Splitting the error equally:

In this case, the error is split equally between all the nodes contributing to that node·



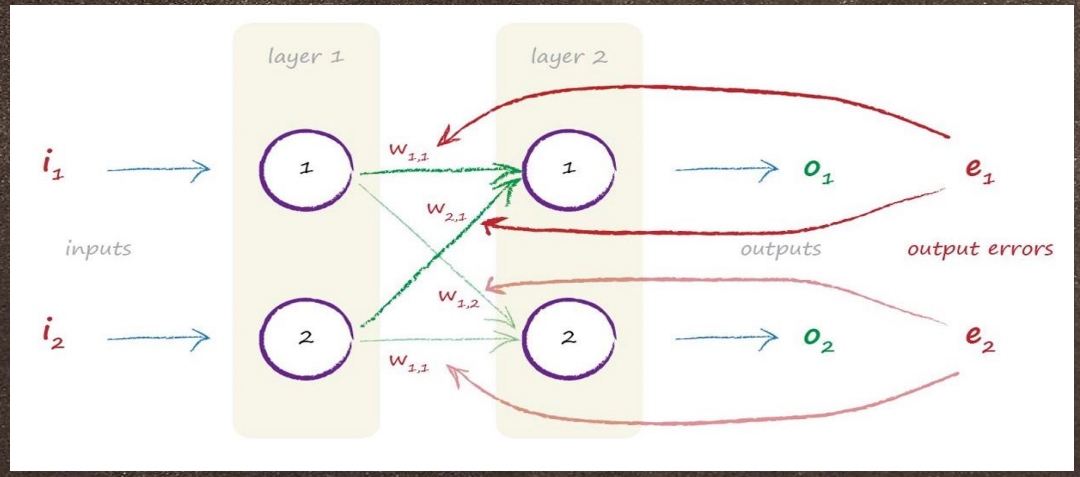
Splitting the error not equally:

The other idea is to split the error but not equally but giving more part to the node which has higher weight because that node contributed more in the error(not true in every case)·



Backpropagating errors from more output nodes:

Now we are going to see how we backpropagate errors from more than one output nodes. In previous slide, we did this for a single output node.
Consider the following neural network with two layers, each layer having two neurons, error from first node is denoted as e_1 while error from the second node is denoted by e_2.



The idea of backpropagating the error will be the same as we discussed earlier for a single node. We will use the technique of giving more parts in error to the node that has a higher weight. We assume that the errors are given as :

$$e_1 = (t_1 - o_1)$$
$$e_2 = (t_2 - o_2)$$

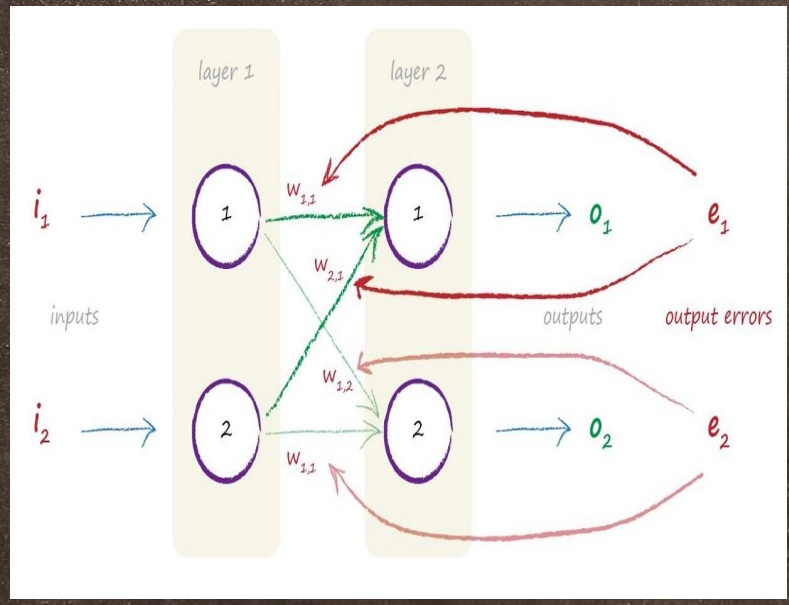
Where t is the training data and o is the actual value.

Backpropagating errors from more output nodes:(cont·)

We can see from the figure that the error e_1 is split in propprtion to the connected links which have weights w_{11} and w_{21} ·Similarly e_2 would be split un proportionate to weights w_{12} and w_{22}

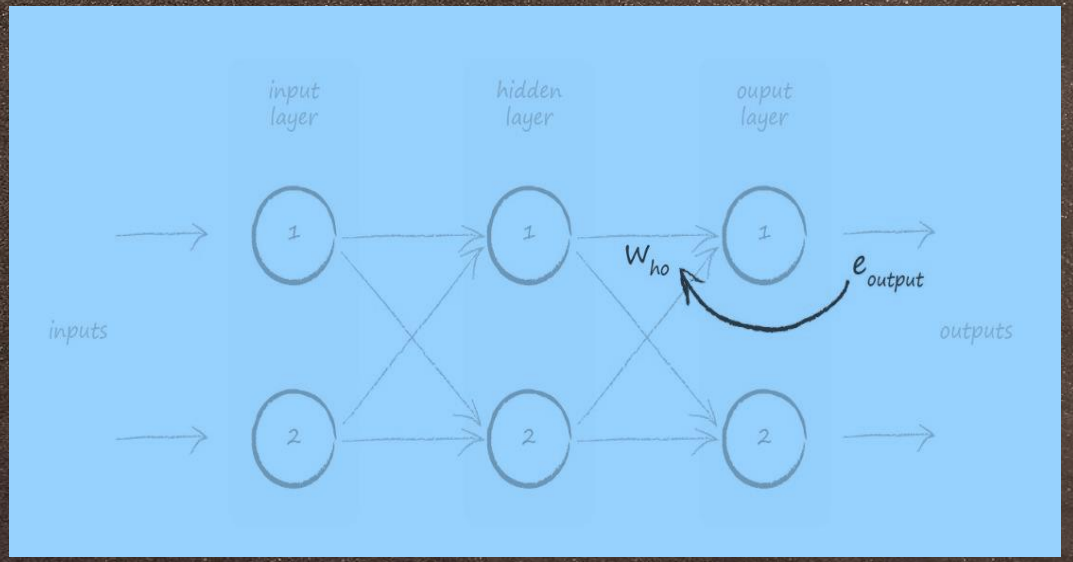
Lets write out what these splits are· The error e_1 is used to inform the refinement of both weights $w_{11}$and w_{21} ·

- To update w_{11}we use the fraction· $\frac{w_{11}}{w_{11}+w_{21}}$
- To update w_{21}we use the fraction· $\frac{w_{21}}{w_{11}+w_{21}}$
- To update w_{12}we use the fraction· $\frac{w_{12}}{w_{12}+w_{22}}$
- To update w_{22}we use the fraction· $\frac{w_{22}}{w_{12}+w_{22}}$



Backpropagating errors to more layers:

The following figure shows a neural network with three layers this time.



This time we need to work with two type of errors. Firstly the output error e_{output} and secondly the hidden error e_{hidden} which is the error in the result produced from hidden layer.

Working with output errors is same as we discussed earlier here we are going to discuss that how can we find the hidden error.

Backpropagating errors to more layers:(cont·)

The question that arises here is that we do not have actual value for the hidden layer as it was for the output layer so the previous technique is not going to work this time· As in the hidden layer we have two neurons and from each neuron two links are emerging one is w_{11} which is associated with e_1 while the second link is w_{12} which is associated with the error e_2 · The error associated with that first node will be the sum of split errors for w_{11} and w_{12}·

$$e_{hidden1} = e_1 \frac{w_{11}}{w_{11} + w_{21}} + e_2 \frac{w_{12}}{w_{12} + w_{22}}$$

Similar work can be done to calculate the error at second node of hidden layer·

$$e_{hidden2} = e_1 \frac{w_{21}}{w_{11} + w_{21}} + e_2 \frac{w_{22}}{w_{12} + w_{22}}$$

Finally we are in position to backpropagate the errors for any neural network does not matter how much layers it has and how much neurons each layer has·

Backpropagating errors with matrix multiplication:

The error from the final layer (output layer) is simply given as:

$$e_{output} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Now remember that the errors for nodes of hidden layer were given as:

$$e_{hidden1} = e_1 \frac{w_{11}}{w_{11} + w_{21}} + e_2 \frac{w_{12}}{w_{12} + w_{22}} \quad \text{and} \quad e_{hidden2} = e_1 \frac{w_{21}}{w_{11} + w_{21}} + e_2 \frac{w_{22}}{w_{12} + w_{22}}$$

Which is nothing but the matrix multiplication given as:

$$error_{hidden} = \begin{pmatrix} \frac{w_{11}}{w_{11} + w_{21}} & \frac{w_{12}}{w_{12} + w_{22}} \\ \frac{w_{21}}{w_{21} + w_{11}} & \frac{w_{22}}{w_{22} + w_{12}} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Malik Ahsin Iqbal

Backpropagating errors with matrix multiplication:(cont·)

Now the last task is to make that weight matrix a bit simpler· As the error was split the way that node having more weight contributed more to the error so in the previous matrix in the fraction the most important and dominant term is the numerator· So for the time being we can ignore the denominator term as it is just the scaling of the errors being backed·

$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

The weight matrix that we got this time is nothing but the transpose of the weight matrix that we constructed at the start so we have reached the final conclusion of this error matrix multiplication·

$$error_{hidden} = W^{T}_{hiddenoutput} \cdot \text{error}_{\text{output}}$$

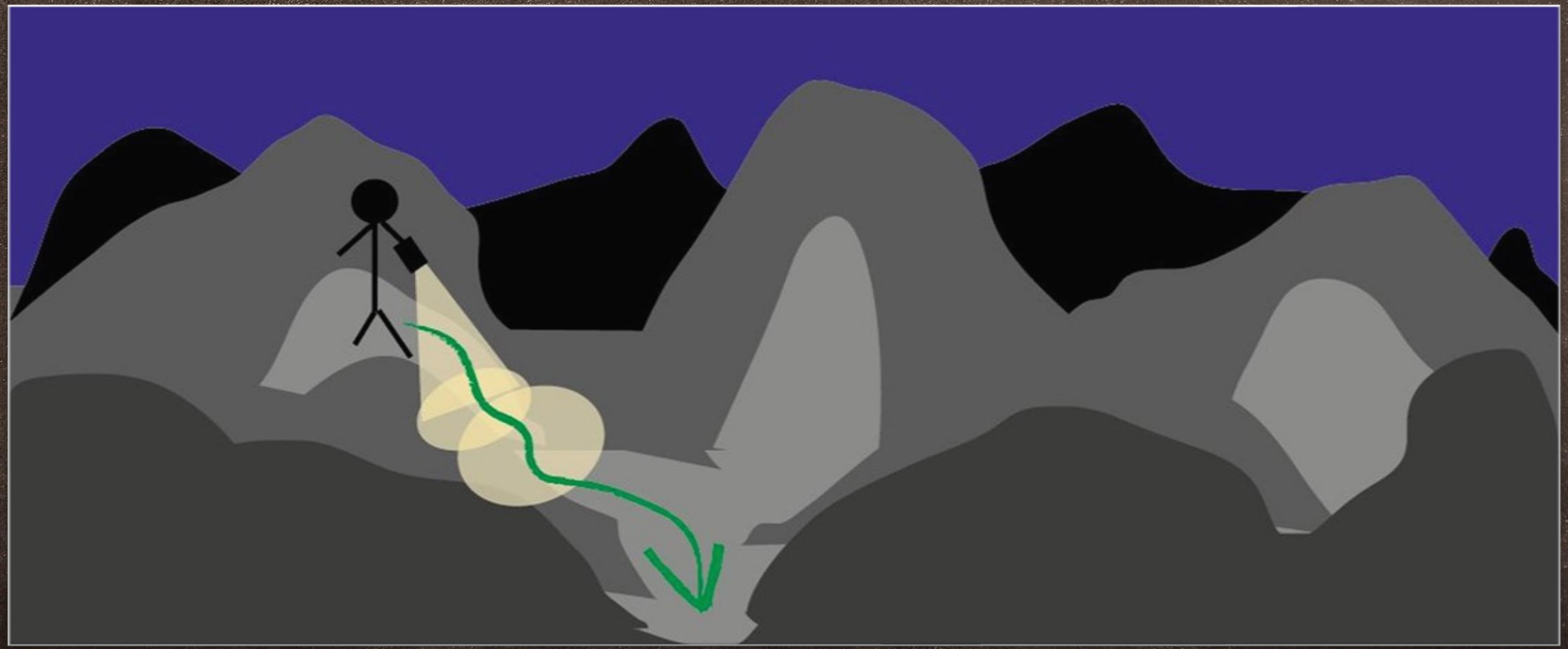
How do we actually update weights?

So far, we've got the errors propagated back to each layer of the network. Why did we do this? Because the error is used to guide how we adjust the link weights to improve the overall answer given by the neural network. This is basically what we were doing way back with the linear classifier in the first part of this presentation.

So we can try an approach called Brute Force. In which we try random combinations of weights to get to target but lets assume that each weight could have 1000 possibilities between -1 to 1. Then for a three layers neural network as we have 18 weights to we have 18000 possible combinations. And what if we have a neural network having 500 nodes in each layer we have 500 million weight possibilities and to try all these combinations if we take 1 second at each combination still it will require 16 years. A thousand training examples and we would be at 16000 years...! So this method is never gonna work.

How do we actually update weights?(cont.)

The technique that we are going to use is called gradient descent. Before going to study this method let us have a look at the following figure:

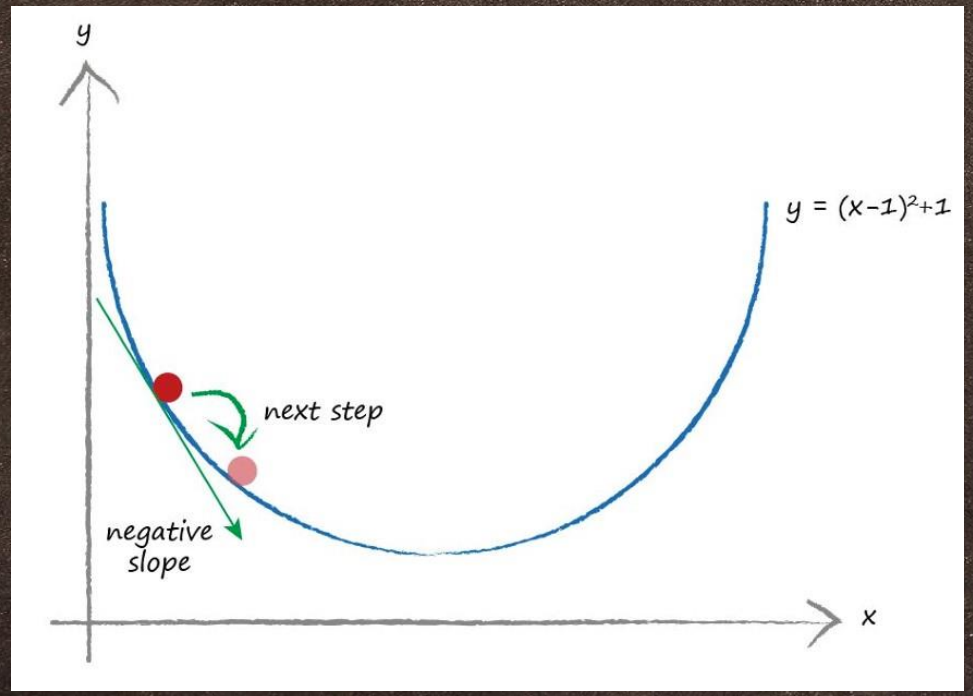


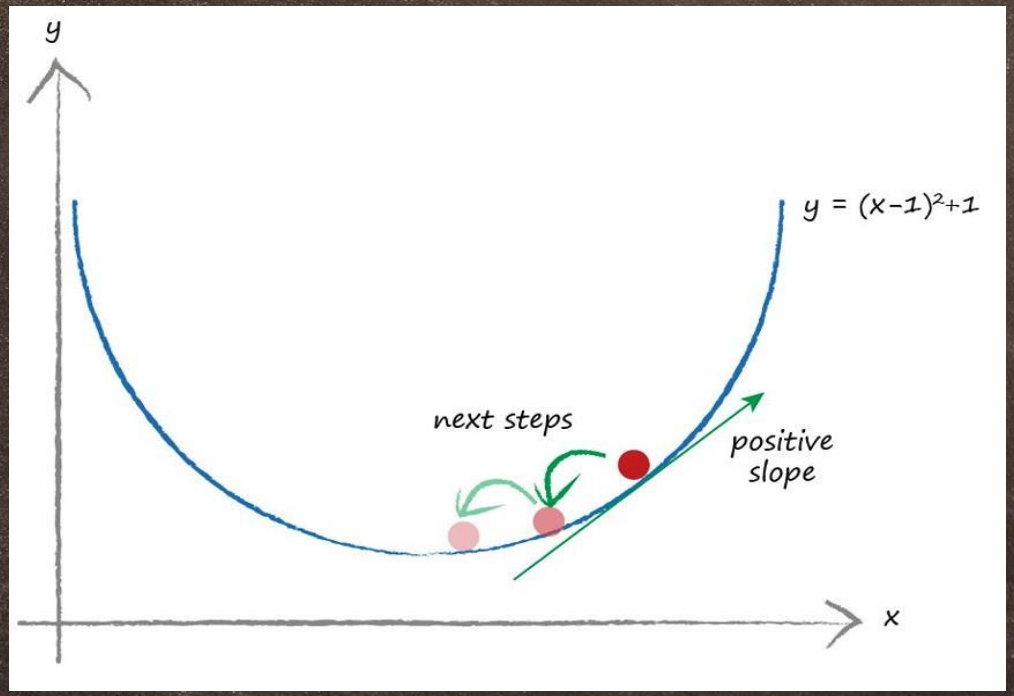
How do we actually update weights?(cont·)

So now we are in position to go mathematically in order to understand gradient descent method·

Look at the following two graphs·

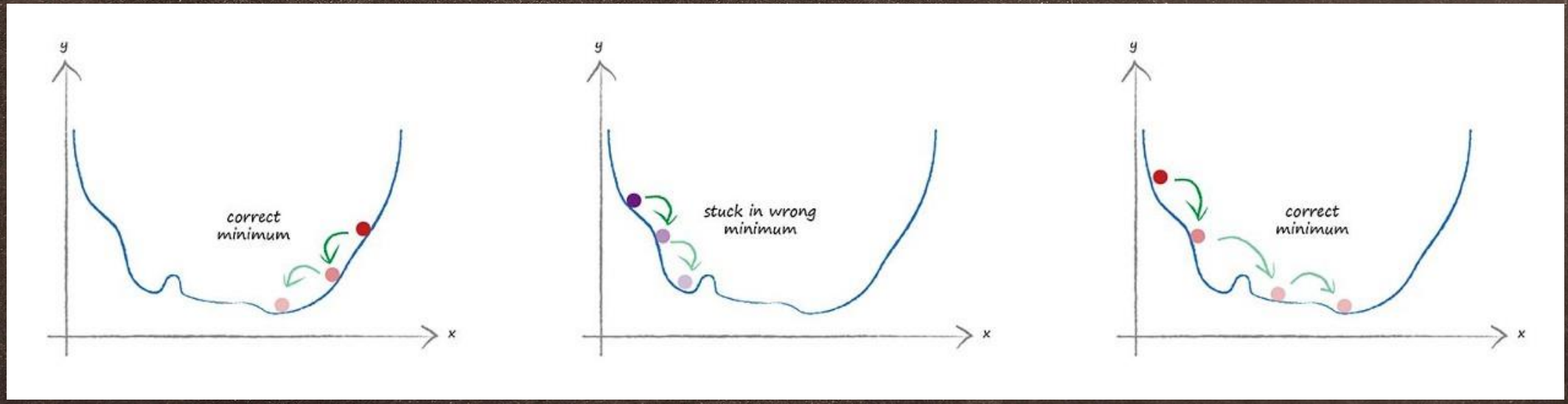
$$y = (x-1)^2 + 1$$





How do we actually update weights?(cont.)

So far so good but our surface will not always be smooth the way it is in previous graphs it can have many ups and downs in it. And when we start walking from the top, it is a possibility that we reach at place which is not the minimum in that whole valley we call it ending up in wrong valley. Just have a look at the following figure.



To avoid this we start updating weights again and again and by changing the initial positions from where we start.

How do we actually update weights?(cont·)

Now lets have a look at the following table in which training values and actuall values for three ouput nodes, together with candidates for an error function are given·

Network Output	Target Output	Error (target - actual)	Error |target - actual|	Error (target - actual)^2
0.4	0.5	0.1	0.1	0.01
0.8	0.7	-0.1	0.1	0.01
1.0	1.0	0	0	0
Sum		0	0.2	0.02

How do we actually update weights?(cont·)

Finally we are in position to do mathematics for Gradient Descent method· As it can be seen in the following figure the steepness is nothing but slope of our error function so we need totake derivative of error function with respect to weights·

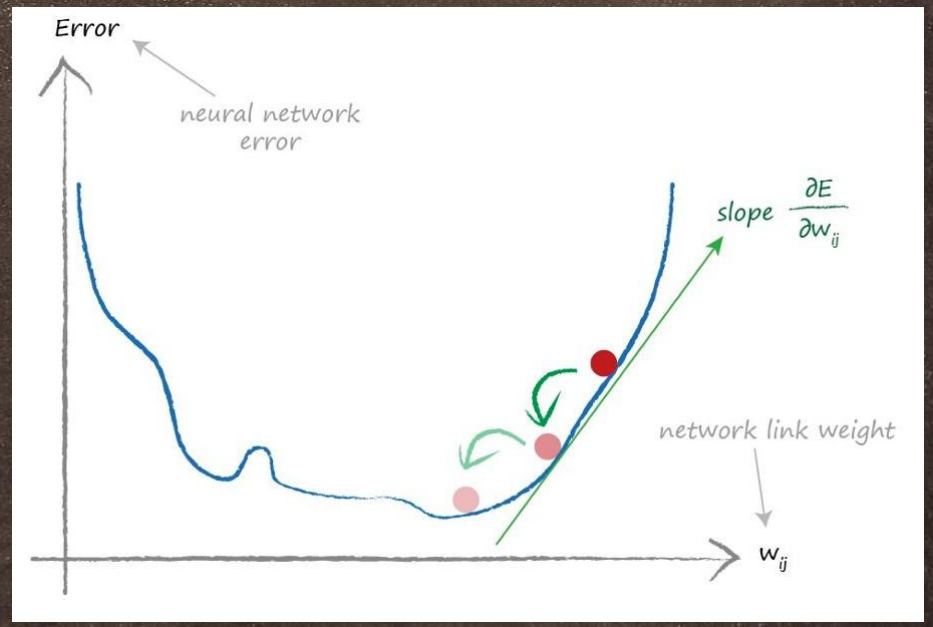
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_{n} (t_n - o_n)^2$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} . \frac{\partial o_k}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) . \frac{\partial o_k}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) . \frac{\partial}{\partial w_{jk}} sigmoid \left(\sum_{j} w_{jk} . o_j \right)$$



How do we actually update weights?(cont·)

Now the derivative of sigmoid function is given ·

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k).\frac{\partial}{\partial w_{jk}} sigmoid\left(\sum_j w_{jk}.o_j\right)$$

$$\frac{\partial}{\partial x} Sigmoid(x) = sigmoid(x)(1 - sigmoid(x))$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k).\ sigmoid\left(\sum_j w_{jk}.o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk}.o_j\right)\right).\frac{\partial}{\partial w_{jk}}\left(\sum_j w_{jk}.o_j\right)$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k).\ sigmoid\left(\sum_j w_{jk}.o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk}.o_j\right)\right).o_j$$

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k).\ sigmoid\left(\sum_j w_{jk}.o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk}.o_j\right)\right).o_j$$

The last expression can be written for input layers and hidden layers as follows·

$$\frac{\partial E}{\partial w_{jk}} = -(e_k).\ sigmoid\left(\sum_j w_{jk}.o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk}.o_j\right)\right).o_j$$

How do we actually update weights?(cont·)

We have completed the main task which was taking the derivative of error function with respect to weights· Now we are going to write the final expression which is used to update the weights·

$$new\ w_{jk} = old\ w_{jk} - \alpha\ .\frac{\partial E}{\partial w_{jk}}$$

The updated weight w_{jk} is the old weight adjusted by the negative of the error slope we just worked out· It's negative because we want to increase the weight if we have a positive slope,and dec rease it if we have a negative slope· The symbol alpha, is a factor
that moderates the strength of these changes to make sure we don't overshoot· It's often called a learning rate·

The matrix form of these weight updated matrices is as follows·

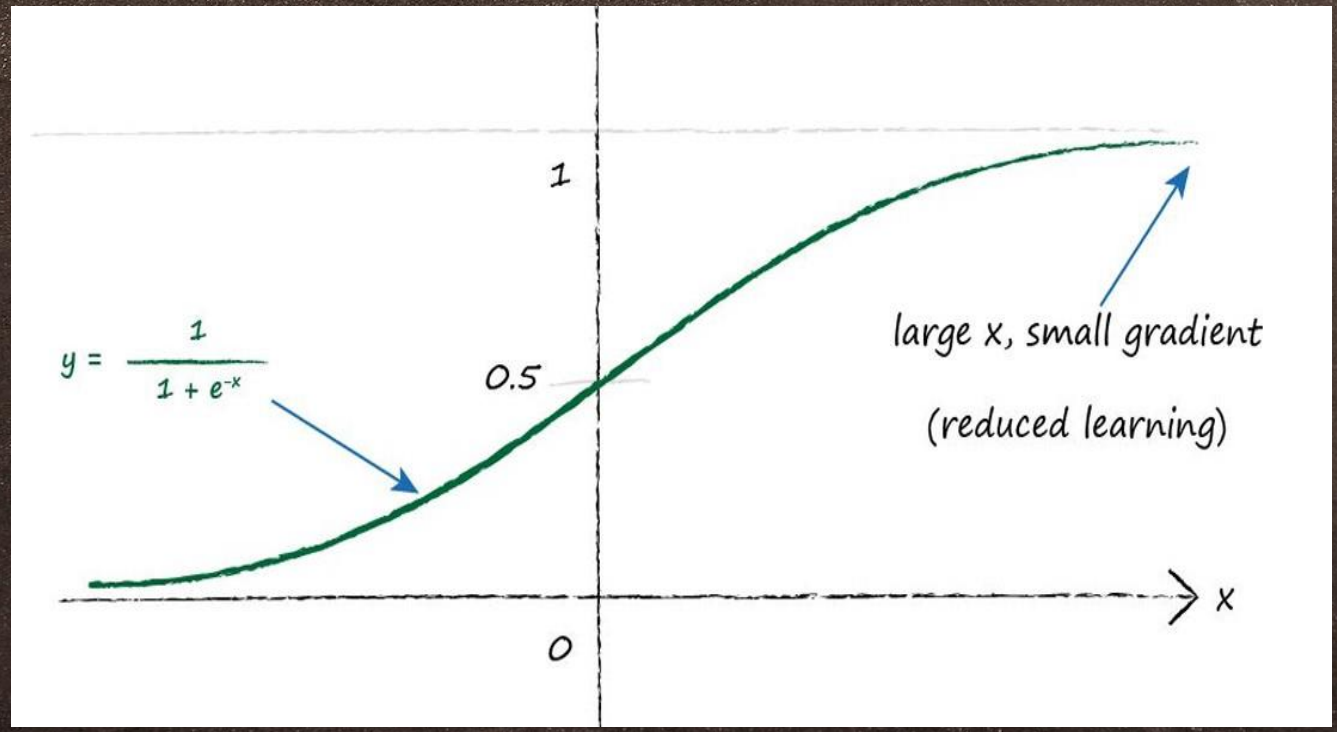
$$\delta W_{jk} = \alpha * E_K * sigmoid\ (O_k) * \left(1 - sigmoid(O_k)\right) * O_i^T$$

Preparing Data:

While preparing the data we must keep the following points in our mind.

1- Inputs:

A common problem is saturation where for large values of inputs our function does not give god results and learning rate decreases. A common range is 0.01 to 0.99 or -1 to 1 depending on the problem. Have a look at the following figure.

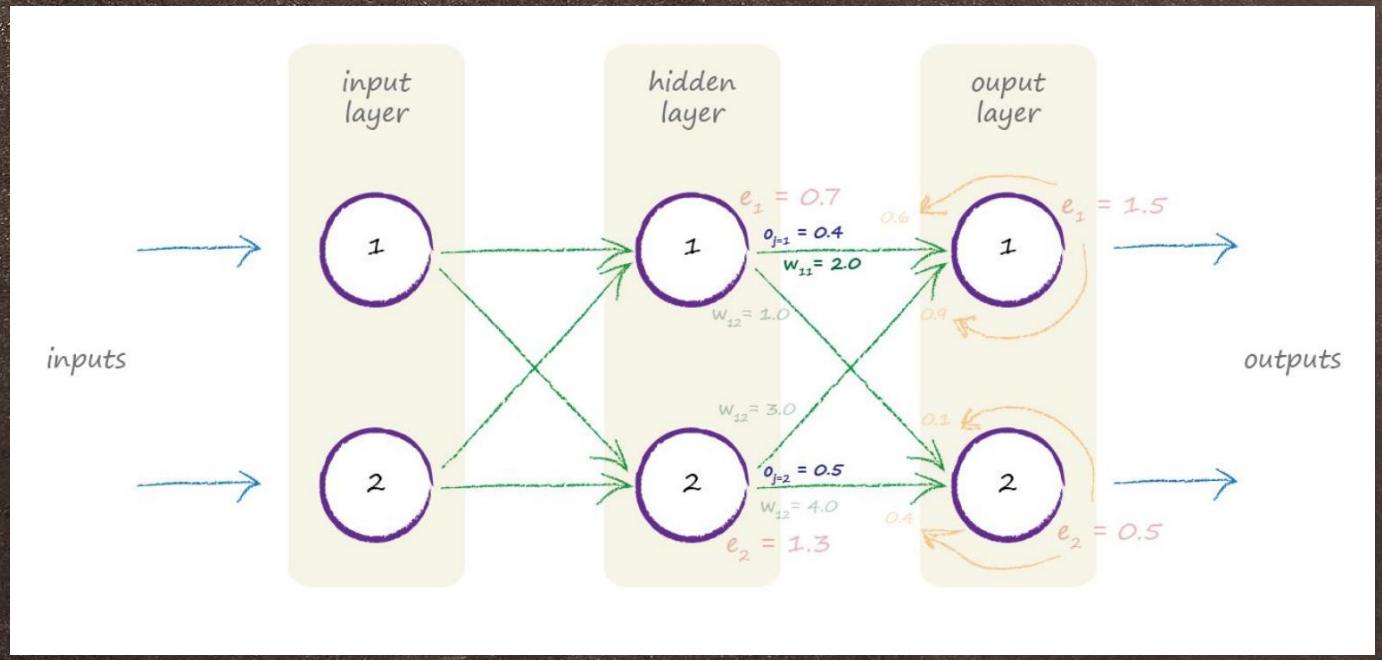


Preparing Data:(cont.)

- Another problem is zero value signals or weights. These also kill the ability to learn better weights.
- The internal link weights should be random and small avoiding zero.
- Output should be in range of what the activation function can produce. Values below 0 or above 1, inclusive, are impossible for the logistic sigmoid. Setting training targets outside the valid range will derive the ever larger weights, leading to saturation.

Practice Problem:

Here a neural network is given with weight values and input output values. Your task is to update the weight by using the above discussed method.



$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) \cdot \text{sigmoid}(\Sigma_j w_{jk} \cdot o_j)(1 - \text{sigmoid}(\Sigma_j w_{jk} \cdot o_j)) \cdot o_j$$

Conclusion:

In this presentation we discussed the structure of neural network from the scratch. All the terminologies associated with neural networks such as input layers, output layers, hidden layers, weight matrix, and updating the weights were discussed in detail.

We also looked at training the model through backpropagation.

THANK YOU FOR YOUR ATTENTION.

QUESTIONS.....?

COMMENTS.....?

Malik Ahsin Iqbal

Acknowledgment:

This presentation was prepared by Malik Ahsin Iqbal and Muhammad Israr Mphil Scholars at ASSMS Batch 22, during the course “Foundations in Deep Learning ” taught by Dr. Jamshaid UL Rahman. These slides cover half of the theoretical part of the book titled “Making your Own Neural Network” by Tariq Rashid.